28장 Number

- 표준 빌트인 객체인 Number는 원시 타입인 숫자를 다룰 때 유용한 프로퍼티와 메서드를 제공한다.

28.1 Number 생성자 함수

- 표준 빌트인 객체인 Number 객체는 생성자 함수 객체
  - 따라서 new 연산자와 함께 호출하여 Number 인스턴스 생성 가능
- Number 생성자 함수에 인수를 전달하지 않고 new 연산자와 함께 호출하면 `[[NumberData]]` 내부 슬롯에 0 을 할당한 Number 래퍼 객체를 생성

```
const numObj = new Number();
console.log(numObj); // Number {[[PrimitiveValue]]: 0}
```

- Number 생성자 함수의 인수로 숫자를 전달하면서 new 연산자와 함께 호출하면 `[[NumberData]]` 내부 슬롯에 인수로 전달받은 숫자를 할당한 Number 래퍼 객체를 생성한다.

```
const numObj = new Number(10);
console.log(numObj); // Number {[[PrimitiveValue]]: 10}
```

- 숫자가 아닌 값을 전달하면 인수를 숫자로 **강제** 변환하고, 변환할 수 없다면 NaN을 `[[NumberData]]` 내부 슬롯에 할당한 Number 래퍼 객체를 생성한다.

```
let numObj = new Number('10');
console.log(numObj); // Number {[[PrimitiveValue]]: 10}

numObj = new Number('Hello');
console.log(numObj); // Number {[[PrimitiveValue]]: NaN}
```

- 명시적으로 타입을 변환하기도 한다.

```
// 문자열 타입 => 숫자 타입
Number('0');     // -> 0
```

```
Number('-1');    // -> -1
Number('10.53'); // -> 10.53

// 불리언 타입 => 숫자 타입
Number(true);  // -> 1
Number(false); // -> 0
```

28.2 Number 프로퍼티

28.2.1 Number.EPSILON

- ES6에서 도입된 Number.EPSILON은 1과 1보다 큰 숫자 중에서 가장 작은 숫자와의 차이와 같다.
- 다음 예제와 같이 부동소수점 산술 연산은 정확한 결과를 기대하기 어렵다.

```
0.1 + 0.2;         // -> 0.30000000000000004
0.1 + 0.2 === 0.3; // -> false
```

- **부동소수점으로 인해 발생하는 오차를 해결하기 위해 사용**

```
function isEqual(a, b){
  // a와 b를 뺀 값의 절대값이 Number.EPSILON보다 작으면 같은 수로
인정한다.
  return Math.abs(a - b) < Number.EPSILON;
}

isEqual(0.1 + 0.2, 0.3); // -> true
```

28.2.2 Number.MAX_VALUE

- Number.MAX_VALUE는 **자바스크립트에서 표현할 수 있는 가장 큰 양수 값**이다.
- Number.MAX_VALUE 보다 큰 숫자는 Infinity다.

```
Number.MAX_VALUE; // -> 1.7976931348623157e+308
Infinity > Number.MAX_VALUE; // -> true
```

28.2.3 Number.MIN_VALUE

- Number.MIN_VALUE 는 **자바스크립트에서 표현할 수 있는 가장 작은 양수 값**이다.
- Number.MIN_VALUE보다 작은 숫자는 0이다.

```
Number.MIN_VALUE; // -> 5e-324
Number.MIN_VALUE > 0; // -> true
```

28.2.4 Number.MAX_SAFE_INTEGER

- 안전하게 표현할 수 있는 가장 큰 정수값이다.

```
Number.MAX_SAFE_INTEGER; // -> 9007199254740991
```

28.2.5 Number.MIN_SAFE_INTEGER

- 안전하게 표현할 수 있는 가장 작은 정수값이다.

```
Number.MIN_SAFE_INTEGER; // -> -9007199254740991
```

28.2.6 Number.POSITIVE_INFINITY

- 양의 무한대를 나타내는 숫자값 Infinity와 같다.

```
Number.POSITIVE_INFINITY; // -> Infinity
```

28.2.7 Number.NEGATIVE_INFINITY

- 음의 무한대를 나타내는 숫자값 -Infinity와 같다.

```
Number.NEGATIVE_INFINITY; // -> -Infinity
```

28.2.8 Number.NaN

- 숫자가 아님을 나타내는 숫자값이다.
- Number.NaN는 window.NaN과 같다.

```
Number.NaN; // -> NaN
```

28.3 Number 메서드

28.3.1 Number.isFinite

- ES6에서 도입된 Number.isFinite 정적 메서드는 인수로 전달된 숫자값이 정상적인 유한수, 즉 Infinity 또는 -Infinity가 아닌지 검사하여 그 결과를 불리언 값으로 반환한다.

```
// 인수가 정상적인 유한수이면 true를 반환한다.
Number.isFinite(0);                // -> true
Number.isFinite(Number.MAX_VALUE); // -> true
Number.isFinite(Number.MIN_VALUE); // -> true

// 인수가 무한수이면 false를 반환한다.
Number.isFinite(Infinity);  // -> false
Number.isFinite(-Infinity); // -> false
```

- 만약 인수가 NaN이면 언제나 false를 반환한다.

```
Number.isFinite(NaN); // -> false
```

- 빌트인 전역함수 isFinite와 차이가 있다.
- **빌트인 전역함수 isFinite**는 전달받은 인수를 암묵적 타입 변환하여 검사를 하지만
- **Number.isFinite**는 전달받은 인수를 숫자로 암묵적 타입 변환을 하지 않는다.

```
// Number.isFinite는 인수를 숫자로 암묵적 타입 변환하지 않는다.
Number.isFinite(null); // -> false

// isFinite는 인수를 숫자로 암묵적 타입 변환한다. null은 0으로 암묵적 타입 변환된다.
isFinite(null); // -> true
```

28.3.2 Number.isInteger

- ES6에서 도입된 Number.isInteger 정적 메서드는 인수로 전달된 숫자값이 정수인지 검사하여 그 결과를 불리언 값으로 반환한다.
- 검사하기전에 인수를 숫자로 암묵적 타입 변환하지 않는다.

```
// 인수가 정수이면 true를 반환한다.
Number.isInteger(0)     // -> true
Number.isInteger(123)   // -> true
Number.isInteger(-123)  // -> true

// 0.5는 정수가 아니다.
Number.isInteger(0.5)   // -> false
// '123'을 숫자로 암묵적 타입 변환하지 않는다.
Number.isInteger('123') // -> false
// false를 숫자로 암묵적 타입 변환하지 않는다.
Number.isInteger(false) // -> false
// Infinity/-Infinity는 정수가 아니다.
Number.isInteger(Infinity)  // -> false
Number.isInteger(-Infinity) // -> false
```

28.3.3 Number.isNaN

- ES6에서 도입된 Number.isNaN 정적 메서드는 인수로 전달된 숫자값이 NaN인지 검사하여 그 결과를 불리언값으로 반환한다.

```
// 인수가 NaN이면 true를 반환한다.
Number.isNaN(NaN); // -> true
```

- **빌트인 전역함수 isNaN은**

  전달단은 인수를 숫자로 암묵적 타입 변환하고,

- **Number.isNaN 메서드는**

  전달받은 인수를 숫자로 암묵적 타입 변환을 하지 않는다.

```
// Number.isNaN은 인수를 숫자로 암묵적 타입 변환하지 않는다.
Number.isNaN(undefined); // -> false

// isFinite는 인수를 숫자로 암묵적 타입 변환한다. undefined는 NaN으
```

```
로 암묵적 타입 변환된다.
isNaN(undefined); // -> true
```

28.3.4 Number.isSateInteger

- ES6에서 도입된 Number.isSateInteger 정적 메서드는 인수로 전달된 숫자 값이 안전한 정수인지 검사하여 그 결과를 불리언 값으로 반환한다.
- 검사전에 인수를 숫자로 암묵적 타입 변환하지 않는다.

```
// 0은 안전한 정수이다.
Number.isSafeInteger(0); // -> true
// 10000000000000000은 안전한 정수이다.
Number.isSafeInteger(10000000000000000); // -> true

// 10000000000000001은 안전하지 않다.
Number.isSafeInteger(10000000000000001); // -> false
// 0.5은 정수가 아니다.
Number.isSafeInteger(0.5); // -> false
// '123'을 숫자로 암묵적 타입 변환하지 않는다.
Number.isSafeInteger('123'); // -> false
// false를 숫자로 암묵적 타입 변환하지 않는다.
Number.isSafeInteger(false); // -> false
// Infinity/-Infinity는 정수가 아니다.
Number.isSafeInteger(Infinity); // -> false
```

28.3.5 Number.prototype.toExponenial

- toExponenial 메서드는 숫자를 지수 표기법으로 반환하여 문자열로 반환한다.

> 지수표기법
>
> 매우 크거나 작은 숫자를 표기할 때 주로 사용하며
>
> e(Exponent) 앞에 있는 숫자에 10의 n승을 곱하는 형식으로 수를 나타내는 방식이다.

- 인수로 소수점 이하로 표현할 자릿수를 전달 할 수 있다.

```
(77.1234).toExponential();  // -> "7.71234e+1"
(77.1234).toExponential(4); // -> "7.7123e+1"
(77.1234).toExponential(2); // -> "7.71e+1"
```

28.3.6 Number.prototype.toFixed

- toFixed 메서드는 숫자를 반올림하여 문자열로 반환한다.
- 반올림하는 소수점 이하 자릿수를 나타내는 0 ~20사이의 정수값을 인수로 전달할 수 있다.
- 인수를 생략하면 기본값 0이 지정된다.

```
// 소수점 이하 반올림. 인수를 생략하면 기본값 0이 지정된다.
(12345.6789).toFixed(); // -> "12346"
// 소수점 이하 1자리수 유효, 나머지 반올림
(12345.6789).toFixed(1); // -> "12345.7"
// 소수점 이하 2자리수 유효, 나머지 반올림
(12345.6789).toFixed(2); // -> "12345.68"
// 소수점 이하 3자리수 유효, 나머지 반올림
(12345.6789).toFixed(3); // -> "12345.679"
```

28.3.7 Number.prototype.toPrecision

- toPrecision 메서드는 인수로 전달받은 전체 자릿수까지 유효하도록 나머지 자릿수를 반올림하여 문자열로 반환한다.
- 인수로 전달받은 전체 자릿수를 표현할 수 없는 경우 지수 표기법으로 결과를 반환한다.
- 전체 자릿수를 나타내는 0~21사이의 정수값을 인수로 전달할 수 있다.
- 인수를 생략하면 기본값 0이 지정된다.

```
// 전체 자리수 유효. 인수를 전달하지 않으면 기본값 0이 전달된다.
(12345.6789).toPrecision(); // -> "12345.6789"
// 전체 1자리수 유효, 나머지 반올림
(12345.6789).toPrecision(1); // -> "1e+4"
// 전체 2자리수 유효, 나머지 반올림
(12345.6789).toPrecision(2); // -> "1.2e+4"
```

```
// 전체 6자리수 유효, 나머지 반올림
(12345.6789).toPrecision(6); // -> "12345.7"
```

28.3.8 Number.prototype.toString

- toString 메서드는 숫자를 문자열로 변환하여 반환한다.
- 진법을 나타내는 2~36 사이의 정수값을 인수로 전달할 수 있다.
- 인수를 생략하면 기본값 10진법이 지정된다.

```
// 인수를 생략하면 10진수 문자열을 반환한다.
(10).toString(); // -> "10"
// 2진수 문자열을 반환한다.
(16).toString(2); // -> "10000"
// 8진수 문자열을 반환한다.
(16).toString(8); // -> "20"
// 16진수 문자열을 반환한다.
(16).toString(16); // -> "10"
```